1. (2) Slojevita izgradnja operacijskih sustava (npr. *arch/kernel/api/programs*) ima svoje prednosti i nedostatke. Navedite neke prednosti i nedostatke.
2. (2) Ako je zastavica `IF` (*Interrupt Flag*) obrisana, hoće li instrukcija `INT 33` generirati prekid (koji će se prihvatiti i obraditi)? Zašto?
3. (2) Opišite „upravljanje“ UI jedinicom koje omogućuje jezgra operacijskog sustava. Koji su sve elementi upravljanja potrebni?
4. (2) Što su to *callback* funkcije? Gdje se one koriste u jezgri operacijskog sustava?
5. Globalne varijable pojednostavljuju komunikaciju između različitih funkcija (nije potrebno prenositi te podatke kao parametre funkcija). Međutim, ponekad one stvaraju probleme.

   a) (2) Neki od njih se manifestiraju kad je kod u više datoteka. Koji su to problemi i kako se rješavaju?

   b) (2) Globalne varijable su osobit problem u višedretvenom okruženju. Koji se problemi ovdje javljaju i kako se rješavaju?
6. (2) Koja je osnovna razlika u komunikaciji preko reda poruka naspram cjevovoda? Skicirati sučelja za oba principa komunikacije.
7. (2) Kako ostvariti zaštitu jezgre od procesa? Opisati mehanizme, mogućnosti te zahtjeve prema sklopovlju.

8. Izvorni kod programske komponente nekog sustava smješten je u direktorije `boot`, `kernel` i `programs`.
   a) (1) Napisati skriptu povezivača koja će pripremiti sustav za početnu adresu 0x10000.
   b) (1) Ponoviti a) ali tako da na početku bude podatkovni dio (konstante i inicijalizirane globalne varijable) iz izvornih kodova iz `boot` direktorija.
   c) (1) Ponoviti a) ali tako da se instrukcije i konstante smjeste na adresu 0x10000 (ROM) a sve ostalo da bude pripremljeno za adresu 0x20000 (RAM), iako će biti početno učitano u ROM iza instrukcija i konstanti.
   d) (2) Ponoviti c) ali tako da su instrukcije i podaci iz `boot` i `kernel` direktorija pripremljeni za apsolutne adrese (navedene u c)) a programi iz `programs` pripremljeni u relativnim adresama (za adresu 0). U skriptu dodati potrebne oznake koje iskoristiti u funkciji `boot()` koja kopira podatke iz ROM-a u RAM, i po kopiranju poziva `startup()`.
9. (3) Neko ugrađeno računalo ima brojilo koje odbrojava od zadane vrijednosti do nule s frekvencijom od 1 MHz. Najveća vrijednost koja stane u brojilo jest `CNT_MAX`. Kada brojilo dođe do nule izaziva prekid broj `CNT_IRQN`, u brojilo automatski učitava `CNT_MAX` te nastavlja s odbrojavanjem. Prekidni podsustav nudi sučelje `void register_interrupt ( irq_num, handler )`. Neka se sadržaj brojila može dohvatiti sa `int cnt_get()` a postaviti sa `void cnt_set (int value)`. Ostvariti sustav praćenja vremena, tj. sučelje `void dohvati_trenutno_vrijeme ( int *sec, int *usec )` koje vraća trenutno vrijeme s preciznošću od mikrosekunde (uz pretpostavku brzog procesora, tj. zanemarenja trajanja izvođenja funkcija za dohvat vremena). Definirati sve potrebne strukture podataka i funkcije.
10. (4) Ostvariti jednostavno raspoređivanje dretvi sučeljem `void schedule ()` koje dretve izravno pozivaju kad žele prepustiti procesor drugim dretvama (ako takvih ima). Pretpostaviti da nema mogućnosti blokiranja dretvi, tj. sve se dretve (osim aktivne) nalaze u redu pripravnih složene prema redu prispijeća (FIFO). Definirati potrebnu strukturu te napisati kod navedene funkcije (i sve ostale pomoćne funkcije). Koristiti C i po potrebi strojne instrukcije (Intel ili ARM arhitekture ili slične onima koje postoje u tim arhitekturama).

11. (4) Ostvariti programsku komponentu za semafor koji prikazuje vrijeme, rezultat te dodatne informacije na zahtjev korisnika. Neka je „uobičajeni“ režim rada: 10 sekundi prikaz trenutnog vremena (minute:sekunde), 5 sekundi prikaz rezultata (domaći:gosti). Na zahtjev (prekid `IRQ_MSG`) treba prikazati traženu poruku (koja se dohvaća s `char *dohvati_poruku ()`) na 15 sekundi te po isteku tih 15 sekundi nastaviti s uobičajenim režimom.
    Na raspolaganju stoje sučelja (NISU jezgrine funkcije!):
    `void registriraj_prekid ( int irqn, void (*funkcija)() )` - za registraciju prekida,
    `void zabrani_prekidanje ()` - za zabranu prihvata prekida naprava,
    `void dozvoli_prekidanje ()` - za dozvolu prihvata prekida naprava,
    `int ispisi ( char *format, ... )` - za ispis („standardna“ *printf* funkcija),
    `void dohvati_poruku ( char *poruka )` - za dohvat poruke koju treba ispisati (na zahtjev),
    `void dohvati_vrijeme ( int *min, int *sek )` - za dohvat trenutnog vremena (koje se nezavisno ažurira – nije potrebno ostvariti, vrijeme kreće od 0:0 na početku igre),
    `void dohvati_rezultat ( int *domaci, int *gosti )` - za dohvat trenutnog rezultata, te
    Korištenjem navedenih sučelja (nema drugih!) ostvariti upravljanje semaforom u funkciji `void semafor ()`. Navesti sve potrebne strukture podataka i pomoćne funkcije. Preciznost vremena i rezultata treba biti unutar sekunde.

12. (4) Ostvariti sustav detekcije potpunog zastoja nad mehanizmom semafora. Pretpostaviti da su sve dretve u listi `lista_t dretve` te da u opisniku dretve `(dretva_t)` postoje elementi: `int stanje` (AKTIVNO, PRIPRAVNO, BLOKIRANO), `lista_t *red` - kazaljka na red u kojem se dretva nalazi. U opisniku semafora `(sem_t)` postoje elementi: `int vrijednost` (trenutna vrijednost semafora) te `lista_t red` – red za blokirane dretve. Svi opisnici semafor nalaze se u listi `lista_t semafori`. Za rad s redom postoji sučelje: `void *red_prvi ( red_t *red )` i `void *red_iduci( void *element )`. Kad je red prazan, odnosno, kad nema idućeg elementa funkcije vraćaju NULL. U funkciji `Čekaj_Semafor ( sem_t *sem )`, nakon blokiranja dretve poziva se funkcija: `provjera_potpunog_zastoja ()` u koju treba dodati kod detekcije potpunog zastoja. Potpuni zastoj definiramo kao stanje kada se SVE dretve nalaze u redu nekog semafora (dretve mogu biti blokirane i zbog drugih razloga, ali tada nije nastupio potpuni zastoj). Pri detekciji potpunog zastoja pozvati `LOG ( ERROR, 'Potpuni zastoj' )`.

13. (4) Prikazati (skicirati) postupak ostvarenja jezgrine funkcije `int povecaj ( int *broj )` koja povećava vrijednost na adresi `broj` za jedan u sustavu u kojem se programi izvode u logičkom adresnom prostoru (adresni prostor procesa kreće od adrese 0). Prikazati i funkciju i pomoćne funkcije i strukture podataka. Odabir načina prijenosa parametara u jezgrinu funkciju je proizvoljan (npr. stog ili zasebne varijable u dretvi).

OSEUP 18.6.2013.

(1.) struktura

```
struct vaprava {
    int *ctr_reg;
    void (*int_hndl)();
};
vap[N] = {NULL, NULL};
```

ϕ

```
int register(void *control_register, void (*interrupt_handler)()) {
    int i;
    for(i=0; i<N; i++) {
        if(vap[i].ctr_reg == NULL) {
            vap[i].ctr_reg = control_register;
            vap[i].int_hndl = interrupt_handler;
            return 0;
        }
    }
    return -1;
}
int unregister(void *control_register) {
    int i;
    for(i=0; i<N; i++) {
        if(vap[i].ctr_reg == control_register) {
            vap[i].ctr_reg = NULL;
            vap[i].int_hndl = NULL;
            return 0;
        }
    }
    return -1;
}
```

```
void int_handler () {
  int i;
  for(i=0; i<N; i++) {
     if(uap[i].cntreg != NULL) {
        if (*uap[i].cntr_reg == 1) {
              uap[i].int_hndl();
              *uap[i].ctr_reg = 0;
        }
  }
}
```

2.

```
int load = 0;
int *adr = 0x1000;
int ms = 250 000 / 1000;

int wdt_start (int wiliseconds) {
    load = wiliseconds
    *adr= load * ms;
    return 0;
}
cut wdt_signal () {
    *adr= load * ms;
    return 0;
}
```

```
int wdt_stop () {
    *adr = 0;
    return 0;
}
```

3. (E)

```
char buf_in [BUF_IN_SZ];
char buf_out [BUF_OUT_SZ];
int in_start, in_sz;
int out_start, out_sz;
#define IRQ 50

int devx_init (){
    in_start = out_start = in_sz = out_sz = 0;
    register_interrupt_handler (IRQ, send_and_receive);
    init (IRQ);
}

int devx_send (void *buffer, size_t size) {
    int i;
    if ( out_sz + size > BUF_OUT_SZ )
        return -1;

    for( i=0; i<size; i++) {
        buf_out [out_start++] = ((char *) buffer) [i];
        if (out_start >= BUF_OUT_SZ)
            out_start = 0;
        out_sz ++;
    }
    if ( get_status () == 0 ) {
        send_and_receive (); // ili sadji ();
    return 0;
}

int devx_recv ....

⇒ vrlo slično!
```

```
void send_and_receive () {
    if ( get_status () != 0)
        return;

    static int zadnje = 0; // zadnje ili primaj, da alternira (ujc neoph.)

    if ( zadnje == 0) {
        salji ();
        primaj ();
    }
    else {
        primaj ();
        salji ();
    }
}

void salji () {
    if (out_sz == 0)
        return;
    if (out_start + out_sz < BUF_OUT_SZ) {
        send (&buf_out[out_start], out_sz);
        out_start += out_sz;
        out_sz = 0;
    }
    else {
        send (&buf_out[out_start], BUF_OUT_SZ - out_start);
        out_sz -= BUF_OUT_SZ - out_start;
        out_start = 0;
    }
}

void primaj () ... slično
```

4. <u>strukture podataka</u>

- opruga zaglavlja, npr:

```
struct header {
    int size;
    struct header *next;
};
```

- lista za slobodne blokove

```
void *malloc(int size) {
    prodi listu slob. blokova i nadi blok odg. veličine
    ako ga nema
        vrati NULL
    inače {
        uzmi blok iz liste
        ako je preveliki podijeli ga na dva dijela
        drugi vrati u listu
        dio dodijeli korisniku => return adresa na zaglavlje
    }
}
void free(void *adr) {
    blok na adr - vel. zaglavlja
    - probaj spojiti sa susjednim sl. blokovima
    - stavi ga u listu slobodnih
}
```

---

5. `#define MAX(A,B,C) \`

```
do {
    int _a = A;
    int _b = B;
    if (_a > _b)
        C = _a;
    else
        C = _b;
} while(0)
```

6. zamijeni_dretve (stara, nova) {

spremi_sve_registre_opće_namjene na stog (ili dretvu)
spremi ~~kontekst stoga~~ registar stanja;
na stog stavi „tu_nastavi";
adresu stoga pohrani u opisnik dretve stara;
odnovi ~~kontekst~~ kontekst stoga iz opisnika dretve nova;
odnovi registar stanja sa stoga
odnovi PC sa stoga (upr. rutr. RET);

tu_nastavi:
vrati se iz funkcije (zamijeni_dretve);

}

---

12. SECTIONS {

. jezgra 0x10000 {

jezgra* (.text);
jezgra* (.rodata);
jezgra* (.data);
jezgra* (.bss);

}

. prog1_ROM . ;

. prog1 0 : AT (. prog1_ROM) {

prog1* (.text .rodata .data .bss)

}

. prog1_size = SIZEOF (. prog1);

. prog2_ROM = . prog1_ROM + . prog1_size;

. prog2 0 : AT (. prog2_ROM)

... slično kao iza prog1 ...

1. (2) Ostvariti prekidni podsustav u sustavu kod kojeg su svi pristupni sklopovi naprava spojeni na isti prekidni signal koji se dovodi do računala. Za svaki pristupni sklop naprave poznata je adresa upravljačkog registra. Čitanjem vrijednosti registra dobiva se stanje pristupnog sklopa: vrijednost 1 označava da je taj pristupni sklop generirao zahtjev za prekid koji črks ns obradu (0 inače). Pisanjem vrijednosti 0 u taj registar javlja se sklopu da je njegov prekid prihvaćen, te će sklop ugasiti svoj zahtjev za prekid (prekidni signal) i nastaviti s radom (i opet generirati zahtjeve za prekid po potrebi). Pisanjem vrijednosti 1 u taj sklop nalaže se sklopu da ne izaziva zahtjeve za prekid (iduća 0 će to opet omogućiti). Prekidni podsustav treba imati sučelja:
   `int register ( void *control_register, void (*interrupt_handler)() );` te
   `int unregister ( void *control_register );`.
   Pored navedenih funkcija prikazati i funkciju `void int_handler ();` koja se poziva pri svakom prekidu (to je već podešeno pri inicijalizaciji sustava), kao i sve dodatne potrebne strukture.

2. (2) Za ostvarenje nadzornog alarma na raspolaganju stoji 32-bitovno brojilo koje odbrojava (frekvencijom od 250 kHz) od zadane vrijednosti do 0, kada izaziva prekid RESET. Upisivanjem vrijednosti u brojilo, ono započinje s odbrojavanjem od te vrijednosti. Međutim, ako se upiše 0 brojilo se deaktivira (ne broji i ne izaziva prekid). Brojilo je dohvatljivo na adresi 0x1000. Napisati slijedeće funkcije za rad s nadzornim alarmom:
   ```
   int wdt_start ( int miliseconds );
   int wdt_stop ();
   int wdt_signal ();
   ```
   Pretpostaviti da pri inicijalizaciji zadajemo vrijeme u milisekundama i da ono neće premašiti vrijednost 1000000.

3. (3) Za neku napravu napisan je skup funkcija:
   ```
   int init ( int irq_num );
   int get_status ();
   int send ( void *buffer, size_t size );
   int recv ( void *buffer, size_t size );
   ```
   Parametar `irq_num` definira koji će prekid naprava izazvati pri primitku novih podataka ili po dovršenju slanja, parametri `buffer` i `size` definiraju međuspremnik za slanje i primanje podataka. Funkcije `send` i `recv` vraćaju veličinu poslanih/primljenih podataka. Funkcija `get_status` vraća 0 ako naprava nema posla, 1 ako je u tijeku slanje ili primanje podataka (napravu treba pustiti da obavi slanje/primanje prije nego li se koriste funkcije `send/recv`). Korištenje ove naprave operacijski sustav treba nuditi kroz sučelje:
   ```
   int devx_init ();
   int devx_send ( void *buffer, size_t size );
   int devx_recv ( void *buffer, size_t size );
   ```
   Pokazati ostvarenje tih funkcija, kao i funkcije koja se poziva na prekid naprave. Pretpostaviti postojanje prekidnog podsustava (funkcije `register_interrupt_handler`). Nadalje, za komunikaciju s napravom rezervirati i koristiti dodatne međuspremnike (ako naprava trenutno već šalje podatke, nove podatke za slanje staviti u međuspremnik koji će se proslijedit napravi kad bude gotova s prethodnim poslom; slično i po čitanju – podaci se prenose u međuspremnik, a iz njega uzimaju sa `recv`).

4. (2) Za ostvarenje dinamičkog upravljanja spremnikom (`malloc/free`) potrebne su neke funkcije i strukture podataka. Skicirajte te funkcije i strukture podataka.

5. (2) Napisati makro `MAX(A,B,C)` koji će veću vrijednost od `A` i `B` staviti u `C` (sam makro ne vraća vrijednost). Pretpostaviti da su parametri tipa `int`, ali i da mogu biti izrazi. Primjeri za koje makro treba raditi ispravno (ali i ne promijeniti semantiku programa!!!):
```
MAX ( 5, a*b, max );
MAX ( a, a+b+c, c );
MAX ( read(f1, b1, n1), read(f2, b2, n2), c );
```
Navedeni pozivi mogu biti nezavisni ili dio `if/else/*` grana, npr.
```
if ( x > 0 )
    MAX (a, b, c );
else
    MAX ( fun1(a), fun2(b), c );.
```

6. (2) Opišite postupak zamjene jedne dretve drugom bez korištenja mehanizma prekida (kako je to napravljeno u `Chapter_07_Threads`).
Skicirajte u pseudokodu ili asembleru funkciju `zamijeni_dretvu ( stara, nova )`.

7. (1) Navedite barem šest elemenata opisnika dretve prema `Chapter_07_Threads` ali opisno (nisu potrebna imena koja se tamo koriste) i za svaki njegovu svrhu.

8. (1) Proširenje sučelja za monitore funkcijom:
`int pthread_mutex_timedlock ( pthread_mutex_t *m, timespec_t *t );`
(dio 3. laboratorijske vježbe) zahtijeva, osim nekih novih funkcija, promjene i u postojećim funkcijama i strukturama podataka. Opišite te promjene.

9. (1) Mehanizam signala omogućava asinkronu "komunikaciju" među dretvama (i OS-a), ali predstavlja i mnoge probleme za ostvarenje jezgre OS-a. Navedite i opišite neke od njih.

10. (2) Kako ostvariti zaštitu (jedne dretve od druge, jezgru od dretvi) u višedretvenom okruženju? Opišite potrebne mehanizme na razini sklopovlja te kako se oni iskorištavaju programski (u jezgri i programima).

11. (1) Pri upravljanju procesima koji koriste logičke adrese javljaju se neki novi problemi (osim korištenja sklopovlja za pretvorbu adresa pri izvođenju procesa). Koji su to problemi i kako se rješavaju?

12. (2) Neki ugrađeni sustav sastoji se od mikro jezgre čiji je kod u direktoriju `jezgra` te tri programa, svaki u svom direktoriju `prog1`, `prog2`, i `prog3`. Ukoliko se programi (procesi) izvode u logičkom adresnom prostoru (koristi se straničenje), napisati zajedničku skriptu za povezivača (linkera) koji će pripremiti jezgru za adresu 0x10000 (u jezgri se koriste fizičke adrese) te programe (koji se učitavaju odmah iza jezgre) za logičku adresu 0 (svaki se zasebno priprema za tu adresu, od svakog programa nastaje zasebni proces). Prilikom pokretanja programa (stvaranja novog procesa), jezgra mora kopirati dio učitanih programa (.data i .bss dijelove) na novu lokaciju. Stoga skripta za povezivača mora uključivati i dodatne varijable.

1. [2 boda] U zaglavlju koje se uključuje iz datoteke `test.c` nalazi se i makro:

```
#define LOG(LEVEL, format, ...) printf("[" #LEVEL ":%s:%d] " format \
"\n", __FILE__, __LINE__, ##__VA_ARGS__)
```

U datoteci `test.c` u liniji 25 makro se poziva sa:

```
LOG ( WARN, "X je izvan granica [10-100]! X=%d", X );
```

Prikazati kako će ta linija izgledati nakon prve faze prevođenja (nakon "preprocesinga", nakon evaluacije makroa).

```
printf("[" "WARN" ":%s:%d] " "X je izvan granica [10-100]! X=%d" \
"\n", "test.c", 25, X);
tj. nakon spajanja konstantnih stringova:
printf("[WARN:%s:%d] X je izvan granica [10-100]! X=%d\n", "test.c", 25, X);
```

2. [3 boda] Sklop za prihvat prekida ima dva registra: KZ (kopija zastavica) i TP (tekući prioritet). Bitovi različiti od nule u registru KZ označavaju da jedna ili više naprava spojenih na taj ulaz traži obradu prekida, dok nule označavaju naprave koje ne traže prekid ili je njihov zahjev prihvaćen (u obradi). Naprava spušta svoj zahtjev kada se pozove funkcija za obradu prekida te naprave. U sustavu ima mnogo naprava i neke (one ista prioriteta) dijele ulaz sklopa za prihvat prekida. Prekidi većeg prioriteta trebaju prekidati obradu prekida manjeg prioriteta. U registru TP bitovi postavljeni u 1 označavaju da je dotični prekid u obradi (procesor postavlja i briše registar TP – to treba ugraditi u kod). Npr. ako je vrijednost KZ $= 00010011_2$ i TP $= 00100100_2$ tada naprave s prioritetima 4, 1 i 0 imaju postavljen zahtjev za prekid dok se trenutno obrađuje zahtjev naprave prioriteta 5, a prekinuta je obrada naprave prioriteta 2 (koja se treba nastaviti po završetku prioritetnijih). Neka postoji funkcija $msb(x)$ koja vraća indeks najznačajnije jedinice (npr. $msb(001000_2) = 3$). Ostvariti prekidni podsustav (`void inicijaliziraj()`, `void registriraj_prekid (int irq, void (*obrada)())` te `void prihvat_prekida()` (bez argumenata!) koja se poziva svaki puta kad se prekid prihvati, za opisani sustav uz pretpostavku da će sklop proslijediti zahtjev većeg prioriteta od tekućeg prema procesoru, dok će one istog ili manjeg prioriteta zadržati (sklop uspoređuje KZ i TP). Pri prihvatu prekida, prije poziva `prihvat_prekida` kontekst prekinuta posla spremljen je na stog, a nakon povratka iz iste funkcije sa stoga se obnavlja kontest (ne treba ga programski spremati/obnavljati). Pretpostaviti postojanje sustava dinamičkog upravljanja gomilom (`kmalloc/kfree`) te operacija za rad s listom (`dodaj_u_listu(&lista,&element)` i `uzmi_prvi_iz_liste(&lista)`).

```
lista_t lista[N];
void inicijaliziraj() {
    int i;
    TP = 0;
    for ( i = 0; i < N; i++ )
        inicijaliziraj_listu ( &lista[i] );
}
void registriraj_prekid (int irq, void (*obrada)()) {
    // registracija NIJE POJAVA prekida !!!
    dodaj_u_listu ( &lista[irq], obrada );
}
void prihvat_prekida() {
    irq = msb (KZ);
    TP = TP | (1<<irq);
    void (*obrada)();
    obrada =  uzmi_prvi_iz_liste ( &lista[irq] );
    while ( obrada ) {
        omogući_prekidanje();
        obrada();
        zabrani_prekidanje();
        obrada =  uzmi_prvi_iz_liste ( &lista[irq] );
    }
    TP = TP ^ (1<<irq);
}
```

3. [3 boda] Neki sustav posjeduje 16 bitovno brojilo na adresi `CNT` koje odbrojava frekvencijom od 50 kHz. Upisom neke vrijednosti u brojilo započinje odbrojavanje prema nuli. Kada brojilo dođe do nule, izaziva prekid te se učitava zadnja upisana vrijednost pa ponovno kreće s odbrojavanjem. Izgraditi sustav upravljanja vremenom koji treba imati sučelja:

   a) inicijalizacija podsustava: `void inicijaliziraj()`
   b) obrada prekida brojila: `void prekid_sata()`
   c) dohvat trenutna sata: `long dohvati_vrijeme()` (vraća vrijeme u ms)
   d) promjena trenutna sata: `void postavi_vrijeme(long novo_vrijeme_ms)`
   e) postavljanje alarma: `void alarm(long za_koliko_ms, void (*obrada)())`

   Sučelje koristi vrijednost sata u milisekundama (pretpostaviti da je tip `long` dovoljan za prikaz sata u milisekundama). Međutim, interno, obzirom da zahtjevi za postavljanjem alarma mogu doći u bilo kojem trenutku, preciznost treba biti veća (u rezoluciji brojila). Promjena sata i postavljanje alarma briše prethodno postavljeni alarm (on se ne poziva). Rješenje neka izaziva prekide što rijeđe, samo kada je to potrebno (kad je to moguće učitavati najveću vrijednost u brojilo). Pretpostaviti da postoje makroi: `OTK_US(O)` koji pretvara broj otkucaja brojila u mikrosekunde i obratno `US_OTK(US)` (vrijeme u mikrosekundama u potreban broj otkucaja brojila).

```
//makroi - nije ih trebalo ostvariti
#define OTK_US(O)     ((O)*20)
#define US_OTK(US)    ((US)/20)

#define MAX_BR 0xffff          //najveći broj koji stane u brojilo
#define MAX_US OTK_US(MAX_BR) //najveći interval između prekida sata
#define MAX_MS (MAX_US/1000)

long sat_ms, sat_us, ucitano, *cnt, odgoda_ms;
void (*alarm)();

void inicijaliziraj () {
    sat_ms = sat_us = odgoda_ms = 0;
    postavi_brojilo ( MAX_MS );
}
void prekid_sata () {
    // pretpostavka je da će se prekid prihvatiti prije nego li brojilo
    // odbroji još koji otkucaj; inače bi trebalo ažurirati drukčije
    azuriraj_sat ();
    if ( odgoda_ms > 0 ) {
        odgoda_ms -= OTK_US ( ucitano ) / 1000;
        if ( odgoda_ms <= 0 ) {
            postavi_brojilo ( MAX_MS );
            alarm();
        }
        else {
            postavi_brojilo ( odgoda_ms );
        }
    }
}
long dohvati_vrijeme () {
    return sat_ms + ( sat_us + OTK_US ( ucitano - (*cnt) ) ) / 1000;
}
void postavi_vrijeme ( long novo_vrijeme_ms ) {
    sat_ms = novo_vrijeme_ms;
    sat_us = odgoda_ms = 0;
    postavi_brojilo ( MAX_MS );
}
```

(nastavak na idućoj stranici)

```
void alarm ( long za_koliko_ms, void (*obrada)() ) {
    azuriraj_sat();
    odgoda_ms = za_koliko_ms;
    postavi_brojilo ( odgoda_ms );
}
void azuriraj_sat () {
    long us = sat_us + OTK_US ( ucitano - (*cnt) );
    sat_ms += us / 1000;
    sat_us = us % 1000;
}
void postavi_brojilo ( long ms ) {
    long usec;
    if ( ms >= MAX_MS )
        ucitano = MAX_BR;
    else
        ucitano = US_OTK ( ms * 1000 );
    *cnt = ucitano;
}
```

4. [3 boda] Izvorni kod neka sustava sastoji se od datoteka u direktorijima: `arch`, `kernel`, `modules` i `programs`. Sliku sustava nastalu prevođenjem treba učitati u ROM na adresi 0x10000. Instrukcije i konstante treba pripremiti tako da ostaju u ROM-u dok ostale varijable treba pripremiti za RAM na adresi 0x30000, osim za programe. Programe pripremiti za logičke adrese, počevši s adresom 0. Obzirom da se koristi straničenje, instrukcije i konstante koje programi koriste moći će se koristiti iz ROM-a, ali će podatke operacijski sustav pri pokretanju programa prekopirati u RAM (na neku slobodnu adresu). Napisati skriptu za povezivača, u koju ugraditi i potrebne varijable, a da bi se kopiranje podataka jezgre i programa moglo obaviti.

```
ROM = 0x10000;
RAM = 0x30000;

SECTIONS {
    .rom1 ROM : AT(ROM) {
        arch* kernel* modules* (.text .rodata)
    }
    adr1 = ROM + SIZEOF(.rom1);
    .rom2 0 : AT(adr1) {
        programs* (.text .rodata)
    }
    adr2 = adr1 + SIZEOF(.rom2);
    .ram1 RAM : AT (adr2) {
        arch* kernel* modules* (.data .bss)
    }
    adr3 = adr2 + SIZEOF(.ram1);
    .ram2 SIZEOF(.rom2) : AT (adr3) {
        programs* (.data .bss)
    }
    adr4 = adr3 + SIZEOF(.ram2);
}
```

5. [3 boda] Neki pisač je spojen na računalo preko pristupna sklopa koji se u računalu upravlja s dva registra: podatkovnim na adresi `POD` te upravljačkim na adresi `UPR`. Spremnost za prihvat podataka pisač iskazuje postavljanjem nule u `UPR`. Bilo koja druga vrijednost označava da pisač trenutno ne može prihvaćati nove podatke za ispis. Napisati upravljački program za pisač koji će u slučaju da pisač ne može prihvaćati nove znakove koristiti i međuspremnik veličine 64 KB. Funkcije koje upravljački program treba ostvariti da bi se one mogle uključiti u sustav su:

```
void *inicijaliziraj (); //vraća opisnik naprave - potrebne podatke
int posalji ( void *podaci, size_t duljina, void *opisnik );
int procitaj ( void *podaci, size_t duljina, void *opisnik );
int dohvati_status ();

#define VMS (64*1024)

struct podaci {
    char ms[VMS];
    size_t prvi, zadnji, ima;
};

void *inicijaliziraj () {
    struct podaci *ms = kmalloc ( sizeof(struct podaci) );
    ms->prvi = ms->zadnji = ms->ima = 0;
    return ms;
}
int posalji ( void *podaci, size_t duljina, void *opisnik ) {
    char *p = podaci;
    size_t jos = duljina;

    //ako ima još nešto u međuspremniku, prvo probaj to slati
    salji_iz_medjuspremnika ( opisnik );

    while ( *UPR == 0 && jos > 0 ) {
        *POD = *p++;
        jos--;
    }

    while ( jos > 0 && op->ima < VMS ) {
        op->ms[op->zadnji] = *p++;
        op->zadnji = ( op->zadnji + 1 ) % VMS;
        op->ima++;
        jos--;
    }

    return duljina - jos; //koliko je poslao i stavio u ms
}
int procitaj ( void *podaci, size_t duljina, void *opisnik ) {
    return 0;
}
int dohvati_status () { return *UPR; }

// ako naprava izaziva prekid, ova funkcija bi se trebala pozvati
void salji_iz_medjuspremnika ( void *opisnik ) {
    struct podaci *op = opisnik;
    while ( *UPR == 0 && op->ima ) {
        *POD = op->ms[op->prvi];
        op->prvi = ( op->prvi + 1 ) % VMS;
        op->ima--;
    }
}
```

6. [3 boda] U nekom sustavu jezgrine funkcije treba pozivati mehanizmom prekida. Ostvariti dio takva sustava i prikazati ga na primjeru poziva `status = ČekajSemafor(s)`. Pokazati ostvarenje te funkcije, ostvarenje prihvata takva prekida i poziv jezgrine funkcije `jf_ČekajSemafor(j_semafor *s)` (prikazanu ispod) i povratak iz jezgrine funkcije u program. Pretpostaviti da postoje funkcije: `izazovi_programski_prekid()`, `dohvati_adresu_prvog_parametara()`, `postavi_povratnu_vrijednost_za_prekinutu_dretvu(vrijednost)` (i ostale koje su bile prikazane u okviru predavanja, ako su potrebne). Pretpostaviti da se struktura `j_semafor` sastoji od vrijednosti semafora i reda za blokirane dretve, dok je `s` tipa `void *` i sadrži fizičku adresu odgovarajuće strukture `j_semafor` (iako se iz procesa toj adresi ne može pristupiti). Dodatno potrebne operacije na "nižoj" razini prikazati pseudokodom (dozvoljeno je "miješanje" pseudokoda i C-a).

```
int jf_ČekajSemafor ( j_semafor *s ) {
    if ( s->vrijednost > 0 ) {
        s->vrijednost--;
    } else {
        blokiraj_trenutnu_dretvu_u_redu(s->red);
        postavi_aktivnu_dretvu()
    }
    return 0;
}

ČekajSemafor(semafor *s) {
    vrati pozovi_j_funkciju ( ČEKAJ_SEMAFOR, s )
}

pozovi_j_funkciju ( int id, ... ) { //u asembleru
    izazovi_programski_prekid() //od tuda ide obrada prekida
    vrati vrijednost iz R0; //prvi registar opće namjene
}

u prihvatu prekida:
    ...
    ako je uzrok_prekida == PROGRAMSKI_PREKID tada
        a = dohvati_adresu_prvog_parametara()
        id = ((int*)a)
        ako je id == ČEKAJ_SEMAFOR tada
            j_semafor *s = a + 1 //idući argument iza id
            pv = jf_ČekajSemafor(s)
            postavi_povratnu_vrijednost(pv)
    ...
```

7. [3 boda] Funkcije iz niže zadanih datoteka pozivaju se i iz drugih datoteka. Opisati što sve ne valja u zadanome kodu. Prepisati te datoteke (ili samo dijelove koje treba promijeniti) ispravnim rješenjem.

```
/* a.c */
# include "sva potrebna zaglavlja su uključna"
int brojac = 0;
static char poruka[] = "12345";
int dodaj ( char *p ) {
    strcat ( poruka, p );   brojac += strlen(poruka) + 1;
    return brojac;
}
/* b.c */
# include "sva potrebna zaglavlja su uključna"
int brojac = 0;
char *u_mala () {
    extern char poruka[]; // varijabla iz datoteke a.c
    int i;
    for ( i = 0; i < strlen(poruka); i++ )
        if ( poruka[i] >= 'A' && poruka[i] <= 'Z' ) {
            poruka[i] = poruka[i] - 'A' + 'a';  brojac++;
        }
    return poruka;
}
int pretvorbi() { return brojac; }
```

```
/* a.c */
# include "sva potrebna zaglavlja su uključna"
static int brojac = 0; //dodan static
char poruka[MAXVEL] = {'1','2','3','4','5',0};
// maknut static, dodana veličina i inicijalizacija
// ili:  char poruka[MAXVEL] = {0}; + kod u dodaj
int dodaj ( char *p ) {
    //if ( poruka[0] == 0 ) strcat ( poruka, "12345" );
    if ( strlen(p) + strlen(poruka) > MAXVEL ) return -1;
    strcat ( poruka, p );   brojac += strlen(poruka) + 1;
    return brojac;
}
/* b.c */
# include "sva potrebna zaglavlja su uključna"
static int brojac = 0;
char *u_mala () {
    extern char poruka[]; // varijabla iz datoteke a.c
    int i;
    for ( i = 0; i < strlen(poruka); i++ )
        if ( poruka[i] >= 'A' && poruka[i] <= 'Z' ) {
            poruka[i] = poruka[i] - 'A' + 'a';  brojac++;
        }
    return poruka;
}
int pretvorbi() { return brojac; }
```

Pisati čitko – nečitak odgovor ne donosi bodove.

1. (1) Navesti prednosti i nedostatke korištenja sustava za upravljanje izvornim kodom (npr. git).

   + praćenje promjena
   + grananje
   + višekorisnički rad
   - složenost (za naučiti)

2. (1) Koja svojstva algoritama dinamičkog upravljanja memorijom treba razmatrati u različitim okruženjima? Koja su bitnija u "običnim operacijskim sustavima", koja u ugrađenim sustavima s vrlo malo memorije, a koja u sustavima za rad u stvarnom vremenu?

   - obični OS: optimirati za prosječnu učinkovitost
   - ugr.sust: fragmentacija, dodatno korištenje memorije za upravljanje
   - SRSV: složenost

3. (1) Koja je prednost korištenja jedinstvenog sučelja za upravljanje napravama – sučelja koje za naprave treba implementirati da bi ih mogli ugraditi u operacijski sustav?

   - prenosivost
   - korištenje više naprava za istu stvar

4. (1) Sinkronizacijske funkcije (npr. semafori) osim uobičajenih sučelja *čekaj* i *postavi*, imaju i dodatna sučelja proširenih mogućnosti. Navesti ta sučelja te opisati njihove proširene mogućnosti.

   - trywait – ne blokiraj ako je semafor* neprolazan
   - timedwait – blokiraj, ali ne beskonačno nego najviše koliko je zadano

5. (1) Ostvarenje jezgrinih funkcija u sustavu koji ima izolaciju preko procesa zahtjeva nekoliko sklopovskih mogućnosti procesora i nekoliko postupaka izgradnje programske potpore. Opišite te potrebne mogućnosti i postupke.

   - potrebne sklopovske mogućnosti:
     * privilegirani način rada (može se sve)
     * korisnički način rada (ograničeni pristupi)
     * prekidi
     * podrška upravljanju spremnikom (segmenti, straničenje)
   - potrebni postupci
     * poziv jezgrinih funkcija mehanizmom prekida
     * zasebna priprema jezgre (npr. za apsolutne adrese)
     * zasebna programa (logičke adrese)

6. (1) Zašto sljedeći makroi nisu dobri? Pokazati primjere kada se ne dobiva željeno ponašanje.

```
#define INV(X)          (1/X)
#define ROOT(a,b,c)     ((-(b)-sqrt((b)*(b)-4*(a)*(c)))/(2*(a)))

a = INV(b+c) => a = 1/b+c
x1 = ROOT(a=funA(a,b,c), b=funB(a,b,c), c--) => a i b se više puta
   koriste u makrou!
```

7. (1) Napraviti makro `PORUKA(TIP,DULJINA,PODACI)` koji će zauzeti memoriju za novu poruku (funkcijom `malloc` koja postoji), popuniti ju zadanim vrijednostima i vratiti kazaljku na nju. Prva dva argumeta makroa su tipa `short`, a treći je adresa međuspremika s podacima. Poruka je definirana strukturom:

```
struct poruka {
```

```
    short tip;
    short duljina;
    char podaci[1];
}

#define PORUKA(TIP, PODACI, DULJINA)                          \
do {                                                          \
   struct poruka *poruka;                                     \
   poruka = malloc ( sizeof(struct poruka) - 1 + DULJINA );   \
   poruka->tip = TIP;                                         \
   poruka->duljina = DULJINA;                                 \
   memcpy ( poruka->podaci, PODACI, DULJINA );                \
   poruka;                                                    \
}                                                             \
while(0)
```

8. (1) Za zadani dio koda navesti gdje će se nalaziti pojedini dijelovi, u kojim odjeljcima (`.text`, `.rodata`, `.data`, `.bss`) ili na stogu.

```
int a = 3;                                      a: .data
static int b = 5;                               b: .data
char rez[10];                                 rez: .bss
void funkcija (int x, int y) {                x, y: stog
    static int z = 0;                           z: .data (.bss)
    char *c = "0123456789";                     c: stog
    ...                                 "0123456789": .rodata
}                                          funkcija: .text
```

9. (3) Neki ugrađeni sustav ima ROM na adresi 0x10000 i RAM na adresi 0x100000. Programe treba pripremiti za učitavanje u ROM, ali za korištenje iz RAM-a – pri pokretanju sve treba najprije kopirati u RAM i tek potom nastaviti s radom. Napisati skriptu za povezivača te funkciju `prekopiraj()` koja će se prva pozvati (staviti ju na početak ROM-a) i napraviti kopiranje te potom pozvati funkciju `pokretanje()` (koja, kao i sve ostalo, treba biti pripremljena za pokretanje iz RAM-a).

```
ldscript.ld:
ROM = 0x10000;
RAM = 0x100000;
ENTRY ( prekopiraj )

SECTIONS {
    .boot ROM :
    {
        prekopiraj.o (*)
    }
    pocetak = ROM + SIZEOF (.boot); //ili samo .
    .sve RAM : AT ( pocetak )
    {
        * (*)
    }
    kraj = pocetak + SIZEOF (.sve);
}

prekopiraj.c:
void prekopiraj() {
```

```
    extern char pocetak, kraj, RAM;
    char *s = &pocetak, *d = RAM;
    for (; s != &kraj;)
        *d++ = *s++;
    pokretanje();
}
```

10. (3) Neki procesor posjeduje sklop za prihvat prekida s N prekidnih ulaza koji se može programirati, tj. zasebno definirati prioritete za svaki pojedini ulaz. Postavljanje prioriteta za pojedini ulaz `I` svodi se na upisivanje prioriteta na adresu `INT_PRIO+I` (npr. za K-ti ulaz treba prioritet upisati na adresu `INT_PRIO+K`). Upisivanjem vrijednosti 0 sklop onemogućava se prekid s zadanog ulaza. Kada se na nekom od ulaza pojavi zahtjev za prekid i taj ulaz ima veći prioritet od broja trenutno zapisanog na adresi `INT_PRIO`, sklop prosljeđuje zahtjev za prekid prema procesoru i u `INT_PRIO` upisuje prioritet novog zahtjeva. Ako novi zahtjev nema veći prioritet sklop neće odmah proslijediti zahtjev za prekid. Kada su prekidi u procesoru omogućeni, procesor pri primitku zahtjeva za prekid sprema kontekst prekinutog programa na stog i poziva funkciju `prihvat_prekida`. Dozvola i zabrana prihvata prekida na razini procesora postiže se instrukcijama `dozvoli_prekidanje` i `zabrani_prekidanje`. Definirati potrebnu strukturu podataka te ostvariti sljedeće funkcije prekidnog podsustava:
`void inicijaliziraj_prekidni_podsustav ();`
`void zabrani_prekide_od_x (int x);` `void dozvoli_prekide_od_x (int x);`
`void registriraj_za_prekid (int x, void (*funkcija)(int), int prio);`
`void prihvat_prekida (int x);` (`x`=ulaz s kojeg je došao zahtjev za prekid).
Obrade prekida treba obavljati prema prioritetima – najprije obraditi prekide većih prioriteta.

```
#define N                (nešto-zadano)
#define INT_PRIO         (nešto-zadano)

void (*obrada[N])(int);      //funkcije za obradu
int *PRIO = (int*) INT_PRIO; //lakše korištenje

void inicijaliziraj_prekidni_podsustav () {
    int i;
    for ( i = 0; i < N; i++ ) {
        obrada[i] = NULL;
        PRIO[i] = 0;
    }
    PRIO[N] = 0;
    dozvoli_prekidanje;
}
void zabrani_prekide_od_x (int x) {
    PRIO[x] = 0;
}
void registriraj_za_prekid (int x, void (*funkcija)(int), int prio) {
    PRIO[x] = prio;
    obrada[x-1] = funkcija;
}
void prihvat_prekida (int x) {
    int tp = PRIO[0]; //zapamti tekući prioritet

    dozvoli_prekidanje;
    obrada[x-1](x);
    zabrani_prekidanje;
    PRIO[0] = tp;
```

```
}
```

11. (3) U nekom sustavu satni mehanizam generira periodičke zahtjeve za prekid svakih 500 mikrosekundi. Ostvariti podsustav za upravljanje vremenom korištenjem samo tog prekida. Jedinica vremena za sva sučelja treba biti milisekunda. Potrebne funkcionalnosti ostvariti kroz sučelja:
`unsigned long int dohvati_sat ();`
`void postavi_sat (unsigned long int t);`
`void alarm (void (*funkcija_za_obradu)(void *), unsigned long int t);.`

```
unsigned long int sat = 0; //sat u jedinicama od 500 mikrosekundi
unsigned long int odgoda = 0;

unsigned long int dohvati_sat () {
    return sat / 2;
}
void postavi_sat (unsigned long int t) {
    sat = t * 2;
}
void alarm (void (*funkcija_za_obradu)(void *), unsigned long int t) {
    odgoda = t * 2;
    obrada = funkcija_za_obradu;
}
void prekid_sata () {
    sat++;
    if (odgoda > 0) {
        odgoda--;
        if ( odgoda <= 0 )
            obrada(NULL);
    }
}
```

12. (2) U nekom sustavu pri pozivu bilo koje jezgrine funkcije prvo se poziva `kthread_pause()`, a neposredno prije povratka iz jezgrine funkcije poziva se `kthread_resume()`. Na poseban alarm koji se poziva svakih `Tq` poziva se i funkcija `kscheduler_tick()`. Definirati potrebnu strukturu podataka (promjene u opisniku dretve) te ostvariti navedene tri funkcije tako da se ostvari raspoređivanje podjelom vremena s kvatnom vremena `Tq`. Dretva treba dobiti najmanje toliko vremena prije nego li će je funkcija `kscheduler_tick()` zamijeniti. Pretpostaviti da za upravljanje dretvama postoje funkcije:
`kthread_t *kthread_get_active_thread ();`
`void kthread_set_active_thread (kthread_t *thread);`
`void kthread_move_to_ready (kthread_t *thread);`
`kthread_t *kthread_get_first_ready ();`
Dohvat trenutnog sata u mikrosekundama obaviti s `long long kget_clock()`.

```
dodati u opisnik kthread_t: long long t_start, t_got;

void kthread_pause() {
    kthread_t *thread = kthread_get_active_thread ();
    thread->t_got += kget_clock() - thread->t_start;
}
void kthread_resume() {
    kthread_t *thread = kthread_get_active_thread ();
    thread->t_start = kget_clock();
}
void kscheduler_tick() {
```

```
    kthread_t *thread = kthread_get_active_thread ();
    if ( thread->t_got >= Tq ) {
        thread->t_got -= Tq;
        kthread_move_to_ready ( thread );
        thread = kthread_get_first_ready();
        kthread_set_active_thread (thread);
    }
}
```

13. (1) Neku jezgru koja je napravljena da radi u sustavu bez zaštite spremnika, mehanizmom procesa treba prilagoditi za takvu zaštitu. Stoga se sve jezgrine funkcije trebaju provjeriti i po potrebi ažurirati. Parametri koje jezgrine funkcije dobivaju su kopije parametara koje je dretva procesa stavila u poziv. U procesima se koriste logičke adrese (svaki proces kreće od adrese 0), a u jezgrinim funkcijama absolutne (fizičke). Početna adresa aktivna procesa (njegova absolutna adresa) u jezgrinim funkcijama se može dohvatiti preko funkcije `kget_process_start()`. Prilagoditi sljedeću jezgrinu funkciju da ispravno radi u tom promijenjenom okruženju.

```
void sys__kernel_funtion ( int uid, char *buffer, int size ) {
    int id = kfind_object ( uid );
    if ( id > 0 ) {
        char *obj = kobj_get_base (id);
        memcpy ( obj, buffer, size );
    }
}
```

```
void sys__kernel_funtion ( int uid, char *buffer, int size ) {
    int id = kfind_object ( uid );
    if ( id > 0 ) {
        char *obj = kobj_get_base (id);
        memcpy ( obj, buffer + kget_process_start(), size );
    }
}
```

(Početni dio za zadatke 1-5) Neki sustav za nadzor i upravljanje ostvaruje se pomoću ugrađenog računala. Izvorni kod nalazi se u direktorijima `kernel` (`boot.c`, `kernel.c`, `interrupts.c`, `time.c`, `net.c` i `startup.c`) i `programs` (`p1.c`, `p2.c` i `p3.c`).

1. (4) Ugrađeno računalo ima ROM na adresi 0x10000 te RAM na adresi 0x100000. Napisati skriptu za prevođenje `ldscript.ld`, kod datoteke `boot.c` te `Makefile`. Instrukcije i konstante pripremiti za učitavanje i korištenje iz ROM-a, a sve ostalo za učitavanje u ROM ali za korištenje iz RAM-a – kod u `boot.c` treba te dijelove kopirati u RAM pri pokretanju te potom pozvati funkciju `start_kernel()`. Pretpostaviti da su u zastavice `CFLAGS`, `LDFLAGS` i `LDLIBS` već postavljene potrebne vrijednosti, a mogu se koristiti i implicitna pravila za određene ciljeve.

```
ldscript.ld
-----------
ROM = 0x10000;
RAM = 0x100000;

SECTIONS {
    .boot ROM : AT (ROM) {
        kernel/boot.o (*)
        * (.text .rodata)
    }
    start = ROM + SIZEOF(.boot);
    .ostalo RAM : AT (start) {
        * (*)
    }
    kraj = start + SIZEOF(.ostalo);
}

boot.c
------
void boot() {
    extern char start, kraj, ROM, RAM;
    size_t i;
    char *a, *b;

    for (a = &start; a < &kraj; a++)
        *a = *b++;

   start_kernel();
}
Makefile
------
PROJECT = project
OBJECTS = boot.o kernel.o interrupts.o time.o net.o startup.o \
p1.o p2.o p3.o

$(PROJECT): $(OBJECTS)
    $(CC) $(LDFLAGS) $OBJECTS -o $(PROJECT) -T ldscript.ld $(LDLIBS)

# teoretski, uz sve zastavice već u *FLAGS dovoljna bi bila i jedna
# linija, npr. boot.elf: boot.o kernel.o ...(itd.)
```

2. (2) Za upravljanje prekidima sustav raspolaže sklopom za prihvat prekida koji ima 8 ulaza: P0-P7. Sklop ima interno dva 8-bitovna registra IR i CP te polje HF[8] (s početkom na adresi 0x4FE2210) u koje treba zapisati adrese funkcije za obradu odgovarajućih prekida. Zahtjev za prekid neke naprave, spojen na neki od ulaza Px, prosljeđuje se u registar IR – odgovarajući bit tog registra se postavlja u 1. Sklop uspoređuje poziciju najviše postavljenog bita iz IR i CP, te ako je najviše postavljeni bit u IR na većoj poziciji od onog u CP te ako je odgovarajuće polje u HF[x] postavljeno (različito od nule) započinje postupak prihvata prekida. U postupku prihvata prekida procesor pohranjuje registre PC i SR na stog te dohvaća adresu iz odgovarajućeg HF[x] u PC (započinje obradu prekida). Istovremeno, sklop za prihvat prekida miče najviši postavljeni bit iz IR u CP. Po završetku obrade prekida, osim što procesor obnavlja PC i SR sa stoga daje signal sklopu za prihvat prekida koji tada briše najviši postavljeni bit iz CP. Ostvariti podsustav za upravljanje prekidima (u `kernel/interrupts.c`) preko sučenja `void init()` i `void register(int irqn, void *hf)`. Korištenjem tog sučelja u funkciji `void start_kernel()` (u `kernel/startup.c`) registrirati funkcije: `time_interrupt()` koji je spojen na P4, `net_interrupt()` koji je spojen na P6 te `sensor_interrupt()` koji je spojen na P7.

```
interrupts.c
------------
void init()
{
    void *HF = (void *) 0x4FE2210;
    int i;
    for (i = 0; i < 8; i++)
        HF[i] = NULL;
}
void register(int irqn, void *hf)
{
    void *HF = (void *) 0x4FE2210;
    HF[irqn] = hf;
}

startup.c
---------
void start_kernel()
{
    register (4, time_interrupt);
    register (6, net_interrupt);
    register (7, sensor_interrupt);
}
```

3. (4) Za upravljanje vremenom na raspolaganju je 24 bitovno brojilo na adresi 0x1CA880 koje frekvencijom od 50 MHz odbrojava od učitane vrijednosti do nule (pa izaziva prekid). Ostvariti podsustav za upravljanje vremenom (sat u `struct timespec {.tv_sec, .tv_nsec}` formatu, jedan alarm) nadopunom datoteke `kernel/time.c`:

```
#define MAXCNT  0xFFFFFF
#define T1NS    ___________ //koliko traje 1 otkucaj u ns - navesti!
#define TIME2NS(X)      ( (X).tv_sec * 1000000000 + (X).tv_nsec )
#define TIME2CNT(X)     ( TIME2NS(X) / T1NS )
#define CNT2TIMENS(X)   ( (X) * T1NS )
#define CNT             ((int*)0x1CA880)
#define SETCNT(X)       do *CNT = (X); while(0)
#define GETCNT()        (*CNT)
static struct timespec clock;
```

```
static struct timespec alarm_delay;
static unsigned last_load;
static void (*alarm_func)();

void time_init() {/* ostvariti */}
void time_set(struct timespec *t) {/* ostvariti */} //... i obr. alarm
struct timespec time_get() {/* ostvariti */}
void time_alarm(struct timespec *t, void (*alarm)()) {/* ostvariti */}
void time_interrupt() {/* ostvariti */}
```

Zanemariti eventualne probleme s preljevom pri množenju cijelih brojeva.

```
...
#define T1NS    20
...
void time_init()
{
    clock.tv_sec = clock.tv_nsec = 0;
    alarm_delay.tv_sec = alarm_delay.tv_nsec = 0;
    alarm_func = NULL;
    last_load = MAXCNT;
    SETCNT(last_load);
}
void time_set(struct timespec *t)
{
    time_init();
    clock = *t;
}
struct timespec time_get()
{
    struct timespec t = clock; //trenutna vrijednost zapisana u satu
    t.tv_nsec += CNT2TIMENS(GETCNT()); //proteklo od ažuriranja sata
    t.tv_sec += t.tv_nsec / 1000000000;
    t.tv_nsec = t.tv_nsec % 1000000000;
    return t;
}
void time_alarm(struct timespec *t, void (*alarm)())
{
    clock.tv_nsec += CNT2TIMENS(last_load - GETCNT());
    clock.tv_sec += t.tv_nsec / 1000000000;
    clock.tv_nsec = t.tv_nsec % 1000000000;

    alarm_delay = *t;
    last_load = TIME2CNT(alarm_delay);
    if (last_load > MAXCNT)
        last_load = MAXCNT;
    SETCNT(last_load);
    alarm_func = alarm;
}
void time_interrupt()
{
    clock.tv_nsec += CNT2TIMENS(last_load);
    clock.tv_sec += t.tv_nsec / 1000000000;
    clock.tv_nsec = t.tv_nsec % 1000000000;
```

```
    if (TIME2NS(alarm_delay) > 0) {
        unsigned remain = TIME2NS(alarm_delay)-CNT2TIMENS(last_load);
        if (remain > 0) {
            alarm_delay.tv_sec = remain / 1000000000;
            alarm_delay.tv_nsec = remain % 1000000000;
            last_load = TIME2CNT(alarm_delay);
            if (last_load > MAXCNT)
                last_load = MAXCNT;
            SETCNT(last_load);
        }
        else {
            last_load = MAXCNT;
            SETCNT(last_load);
            alarm_func();
        }
    }
}
```

4. (4) Za komunikaciju s drugim uređajima ugrađeno računalo ima mrežnu karticu. Kada stigne novi paket preko mreže, on se korištenjem izravnog pristupa spremniku (DMA) prebacuje u radni spremnik na adresu koja treba biti zapisana u mrežnu karticu na adresi 0x72F4180 (maksimalna veličina paketa je 128 B). Po prebacivanju paketa mrežna kartica izaziva prekid i upisuje veličinu paketa na adresu 0x72F4184. Tek po obradi prekida i upisivanja vrijednosti nula na adresu 0x72F4184 mrežna kartica može zaprimati iduće pakete s mreže. Slanje paketa može se obaviti tek ako je na adresi 0x72F4188 nula i to tako da se na adresu 0x72F418C upiše duljina paketa u oktetima te na adresu 0x72F4190 upiše početna adresa podataka u spremniku. Naprava će kraj slanja dojaviti prekidom i postavljanjem nule na adresi 0x72F4188 (tek tad se može osloboditi memorija u kojoj je paket). Korištenjem opisane mrežne kartice ostvariti podsustav za upravljanje mrežom koji će dodatno koristiti dva reda poruka, jedan za dolazne pakete a jedan za odlazne. Potrebno sučelje koje treba ostvariti u `kernel/net.c` jest:

```
//već definirano (npr. u include/net.h)
struct packet {size_t size; char data[1];};
//samo se "data" dio šalje/prima preko mreže
//ostvariti u kernel/net.c
void net_init() {}
void net_send(struct packet *packet) {}
int net_recv(struct packet **packet) {} //vraća veličinu primljena
//paketa, u packet stavlja adresu primljena paketa koju net_recv treba
//rezervirati s kmalloc
void net_interrupt() {}
```

Pretpostaviti da na na raspolaganju stoje operacije `memcpy`, `kmalloc`, `kfree` te operacije za rad s redom poruka: `id = mq_create()`, `void mq_send(id, void *data)`, `int mq_recv (id, void **data)` koje rade s kazaljkama (u red stavljaju i vraćaju kazaljku, ne kopiraju podatke!).

```
struct packet {size_t size, char data[1]};
static id_r, id_s;
static char input_buffer[128];

void net_init()
{
    id_r = mq_create();
```

```
    id_s = mq_create();
    *((int*)0x72F4180) = input_buffer;//kamo da sprema dolazeći paket
}
void net_send(struct packet *packet)
{
    if (*((int*)0x72F4188) == 0) {
        *((int*)0x72F418C)  = packet->size;
        *((int*)0x72F4190)  = packet->data;
    }

    //u ovom rješenju uvijek se "zapamti" paket, čak i ako je stavljen
    //za slanje; na prekid, kad je paket poslan, onda se miče i briše
    mq_send(id_s, packet);
}
int net_recv(struct packet **packet)
{
    return mq_recv(id_r, &packet);
}
void net_interrupt()
{
    size_t size = *((int*)0x72F4184);
    if (size) {//ima novog
        struct packet *p = kmalloc(sizeof(struct packet)+size);
        p->size = size;
        memcpy(p->data, input_buffer, size);
        mq_send(id_r, p);
    }
    if (*((int*)0x72F4188) == 0) {//spreman za (novo) slanje
        struct packet *packet;
        if ( mq_recv(id_s, &packet) ) {//prvi u redu je poslan
            kfree(packet); //sad ga obriši
            if ( mq_recv(id_s, &packet) ) { //ima li još?
                *((int*)0x72F418C)  = packet->size;
                *((int*)0x72F4190)  = packet->data;
            }
        }
    }
}
```

5. (3) U programu `p1.c` ostvariti funkciju `sensor_interrupt()` koja će na prekid senzora poslati poruku sadržaja "PING" (korištenjem opisanog sučelja `net_send` u prethodnom zadatku). Program u `p2.c` svakih 5 sekundi treba provjeriti ima li pristiglih poruka i ako ima njihov sadržaj poslati na LCD upisivanjem njena sadržaja na adresu 0x328910 (koristiti ostavarena sučelja za rad s vremenom). Program u `p3.c` u petlji učitava vrijednost s treće naprave, s adrese 0x9871230 te ako je pročitana vrijednost veća od 90 šalje poruku "HOT" preko mreže te na LCD ispisuje "FLUID TOO HOT". Program u `p2.c` se pokreće samoj jednom (jer se njegova funkcionalnost stavlja u alarm), dok program u `p3.c` radi cijelo vrijeme (tj. prekida se samo prekidima mreže, senzora, brojila).

`p1.c`:

```
void sensor_interrupt()
{
    struct packet *p = kmalloc(sizeof(struct packet)+4);
    p->size = 4;
```

```
    memcpy(p->data, "PING", 4);
    net_send(p);
}
p2.c:
void init_alarm()
{
    struct timespec t;
    t.tv_sec = 5;
    t.tv_nsec = 0;
    time_alarm(&t, alarm);
}
void alarm()
{
    struct packet *p;
    if (net_recv(&p)) {
        size_t i;
        for (i = 0; i < p->size; i++)
            *((int*)0x328910)   = packet->data[i];
            //ovdje ne može memcpy jer on mijenja obje adrese!
    }
}
p3.c:
void p3()
{
    while(1) {
        if (*((int*)0x9871230) > 90) {
            struct packet *p = kmalloc(sizeof(struct packet)+3);
            p->size = 3;
            memcpy(p->data, "HOT", 3);
            net_send(p);

            size_t i;
            char msg[] = "FLUID TOO HOT";
            for (i = 0; i < 13; i++)
                *((int*)0x328910)   = msg[i]; //ne memcpy!
        }
    }
}
```

6. (3) U nekom sustavu koji koristi virtualni spremnik (procesi se nalaze u logičkim adresama) nalazi se jezgrina funkcija `int ktest_strings(char **array)`. Zadatak je funkcije zasebno provjeravati nizove znakova (string) iz polja nizova `array` već postojećom funkcijom `int ktest_string(char *string)`. Zadnji element polja ima vrijednost `NULL` (veličina polja nije drukčije zadana). Kazaljka `array` kao i njeni elementi su u logičkim adresama. Pretpostaviti da postoje funkcije `void *adr_u2k(void *logical_address)` i `void *adr_k2u(void *physical_address)` koje pretvaraju logičku u fizičku adresu i obratno (za trenutno aktivan proces). Funkcija `int ktest_strings` treba vratiti 0 ako su sve povratne vrijednosti od `ktest_string` također bile nule. U protivnom potrebno je vratiti indeks prvog elementa iz polja `array` za koji nije dobivena nula.

```
int ktest_strings(char **array)
{
    char **ka = adr_u2k(array); //u fizičku - adresa početka polja s
```

```
                                    //kazaljkama
    char *string;
    int i;

    for (i = 0; ka[i] != NULL; i++) {
        string = adr_u2k(ka[i]); //svaku kazaljku opet u fizičku adr.
        if (ktest_string(string) != 0)
            return i;
    }
    return 0;
}
```

Završni ispit iz predmeta Operacijski sustavi za ugrađena računala
30. 6. 2021. Student: _____

1. (4) Programska komponenta priprema se za sustav koji ima ROM na adresi 0x10000 te RAM na adresi 0xA0000. Napisati skriptu za povezivanje koja treba generirati sliku sustava koja će se posebnim alatom učitati u ROM. Sve što može ostati u ROM-u neka tamo trajno ostane i od tuda koristi, a sve ostalo pripremiti za korištenje iz RAM-a. Napisati funkciju *premjesti()* koja se poziva odmah pri pokretanju sustava a koja kopira potrebne dijelove iz ROM-a u RAM. Po potrebi koristiti i postojeću funkciju void `void memcpy(void *dest, void *src, size_t size)`.

Rješenje:

```
ROM = 0x10000
RAM = 0xA0000
SECTIONS
{
   .text ROM : {
      * (.text)
      * (.rodata)
   }
   text_size = SIZEOF(.text);
   data_in_rom = .;
   .data RAM : AT(ROM + text_size) {
      * (.data)
      * (.bss)
   }
   data_size = SIZEOF(.data);
}
```

```
void premjesti()
{
   extern char RAM, data_in_rom, data_size;
   memcpy(&RAM, &data_in_rom, &data_size);
}
```

2. (4) Neki sustav ima sklop za prihvat prekida s 16 ulaza na koje se spajaju naprave kada traže prekid. Sklop ima sljedeće registre: 16-bitovni upravljački registar (`UR`) na adresi `0xFB00`, 8-bitovni registar prioriteta (`RP`) na adresi `0xFB04` te 8-bitovni registar trenutna prioriteta (`TP`) na adresi `0xFB08`. Svaki bit upravljačkog registra označava da li se razmatra ili ne ekvivalentni ulaz – kada je bit obrisan (nula) zahtjev za prekid dotične naprave se ignorira, u protivnom se prihvaća. Svaki ulaz ima dodijeljeni prioritet koji odgovara rednom broju ulaza: ulaz 0 ima prioritet 0 (najmanji prioritet), ulaz 15 prioritet 15 (najveći prioritet). U registru `RP` nalazi se prioritet najprioritetnijeg zahtjeva koji čeka na obradu ili broj `0xFF` kada nema niti jednog zahtjeva. Ako je vrijednost u `RP` veća od one u `TP`, sklop šalje zahtjev za prekid prema procesoru. Pri prihvatu prekida procesor treba upisati prioritet trenutne obrade u `TP`. Sklop će tada obrisati taj zahtjev za prekid u svojim internim registrima i po potrebi ažurirati registar `RP`. Ostvariti podsustav za upravljanje prekidima (u C-u) sa sučeljima za inicijalizaciju podsustava, funkciju za registraciju funkcije za obradu prekida te funkciju koja se poziva na svaki prekid (a koja mora utvrditi prioritet prekida i pozvati odgovarajuću registriranu funkciju). Započetu obradu prekida ne prekidati novim zahtjevima, makar i većeg prioriteta (obradu obaviti sa zabranjenim prekidanjem). Onemogućiti ulaze za koje ne postoji funkcija za obradu (tako da se takvi zahtjevi ne prosljeđuju procesoru).

Rješenje:

```
short int *UR = (short int *) 0xFB00;
unsigned char *RP = (unsigned char *) 0xFB04;
unsigned char *TP = (unsigned char *) 0xFB08;
void (*obrada[16])();

void inicijaliziraj_prekide()
{
   int i;
   for (i = 0; i < 16; i++)
      obrada[i] = NULL;
   *TP = 0;
   *UR = 0;
}
```

```
void registriraj(int prio, void *funkcija)
{
   obrada[prio] = funkcija;
   if (funkcija != NULL)
      *UR = *UR | (1<<prio);
   else
      *UR = *UR & ~(1<<prio);
}
void prekidna_rutina() //vidi komentar ispod
{
   int prio = *RP;
   *TP = prio;
   obrada[prio]();
   *TP = 0;
}
```

Samo zahtjevi onih naprava za koje su postavljene funkcije mogu se proslijediti dalje (ovdje), pa nije potrebno ispitivati da li je ta funkcija postavljena. Zabrana prekida i dozvola nije potrebna jer procesor pri prihvatu prekida automatski zabrani daljnje prekidanje, a pri povratku iz prekida dozvoli ponovno prekidanje.

3. (4) Neki sustav ima brojilo na adresi `BROJILO` koje odbrojava frekvencijom f=800 kHz od učitane vrijednosti do nule kada izaziva prekid te postavlja najveću vrijednost u brojilo (`0xFFFF`) i nastavlja s odbrojavanjem (ciklus ima najviše `0xFFFF` otkucaja). Korištenjem navedena brojila ostvariti podsustav za upravljanje vremenom koji se sastoji od sata izraženog u mikrosekundama (tip `long` u C-u), te jednog alarma sa sučeljima:

```
void inicijaliziraj();
void dohvati_sat (long *t);
void postavi_sat (long *t);
void postavi_alarm (long *kada, void (*alarm)());
void prekid_sata();
```

Argument `kada` je apsolutno vrijeme izraženo u mikrosekudama. Pri promjeni sata funkcijom `postavi_sat`, obrisati alarm bez njegove aktivacije. Za pretvorbu broja otkucaja u vrijeme i obratno napraviti makroe `CNT2USEC(CNT)` i `USEC2CNT(USEC)`. Pretpostaviti da vrijeme izraženo u mikrosekundama neće preći najveću vrijednost koja stane u tip `long` (tj. vrijednost `MAXLONG`).

Rješenje:

```
#define MAXCNT 0xFFFF
#define FREQ 800000 //ne koristi se

#define CNT2USEC(CNT) ((CNT)*5/4) //kom.isp.
#define USEC2CNT(USEC) ((USEC)*4/5)

static long sat;
static void (*obrada_alarma)();
static long t_akt;
static unsigned short int ucitano;
static unsigned short int *brojilo = BROJILO;

static void podesi_brojilo()
{
   if (t_akt < MAXLONG) {
      long za_koliko = USEC2CNT(t_akt - sat);
      if (za_koliko > 0 && za_koliko < MAXCNT)
         ucitano = za_koliko;
   }
   else {
      ucitano = MAXCNT;
   }
   *brojilo = ucitano;
}

void inicijaliziraj()
{
   sat = 0;
   obrada_alarma = NULL;
   t_akt = MAXLONG;
   podesi_brojilo();
}
```

```
void dohvati_sat(long *t)
{
   *t = sat + CNT2USEC(ucitano - *brojilo);
   //Ovdje ne ažurirati sat! Jer na slijedeći prekid će se opet isto
   //vrijeme nadodati na sat.
}

void postavi_sat(long *t)
{
   inicijaliziraj();
   sat = *t;
}

void postavi_alarm (long *kada, void *alarm)
{
   sat += CNT2USEC(ucitano - *brojilo);
   t_akt = kada;
   obrada_alarma = alarm;
   podesi_brojilo();
}

void prekid_sata ()
{
   sat += CNT2USEC(ucitano);
   if (t_akt <= sat) {
       t_akt = MAXLONG;
       podesi_brojilo();
       obrada_alarma();
   }
   else {
      podesi_brojilo();
   }
}
```

Ostaviti „*1000000/800000“ nije dobro jer se prvim množenjem može izaći iz opsega broja tipa long (najčešće 32 bita). Obzirom da su brojke okrugle to se lako pokrati i ostane 5/4.

CNT je cijeli broj i množiti ovo realnim brojem 1,25 nije preporučeno, pogotovo što je očito ovo program za neki mikrokontroler, a oni gotovo uvijek nemaju operacije s realnim brojevima.

Nadalje, bitno je prvo pomnožiti s 5 a onda podijeliti s 4. Obrnutim redoslijedom se gubi na preciznosti.

4. (4) Računalo nekom udaljenom uređaju šalje podatke preko posebnog sklopa sljedećim protokolom: prvo se pošalje bajt `0xF0` koji označava početak poruke; slijedi bajt koji označava duljinu korisne informacije koja se želi poslati; potom se šalje bajt po bajt informacije te na kraju se još pošalje bajt `0x0F`. Sklop koji se koristi za slanje ima dva registra: registar podatka `RP` preko kojeg se šalju podaci te registar stanja (`RS`). Prvi bit registra stanja (jedini bit koji se koristi u tom registru) označava je li sklop spreman prihvatiti sljedeći bajt za slanje. Pri pisanju u `RP` se taj bit automatski obriše dok sklop ne bude spreman za prihvat novog bajta, kada ga sklop ponovno postavlja. Svaki puta kada se taj bit u postavi u 1 sklop izaziva zahtjev za prekid na koji se poziva funkcija `obrada_prekida_x()` (koju treba ostvariti). Ostvariti upravljanje navedenim sklopom uz poštivanje opisanog protokola kroz funkciju:

   `int pošalji(void *informacija, char veličina);`

   gdje `informacija` sadrži informaciju koju treba poslati (bez dodatnih bajtova protokola) te `veličina` njenu veličinu. Funkcija treba raditi asinkrono, tj. postaviti zadanu poruku za slanje, eventualno odraditi i početno slanje, ali se ostatak slanja treba ostvariti u funkciji za obradu prekida. Funkcija `pošalji` treba vratiti nulu kad je zadala slanje. Ako se `pošalji` pozove dok prethodna operacija nije gotova poziv treba vratiti `-1` i taj se novi zahtjev ignorira. Ostvariti funkcije `inicijaliziraj`, `pošalji` i `obrada_prekida_x` (uz potrebnu dodatnu strukturu podataka za privremenu pohranu poruke).

   Rješenje:

```
char duljina, sljedeći;
char bufer[127]; //može i 255; problem je u „char veličina“ {-128 do 127}
int status; // 0 - ništa, 1 - slanje u tijeku

void inicijaliziraj ()
{
   status = 0;
}

int pošalji (void *informacija, char veličina)
{
   if (status != 0)
      return -1;
   status = 1;
   duljina = veličina + 3;
   sljedeći = 0;
   bufer[0] = 0xF0;
   bufer[1] = veličina;
   memcpy(bufer + 2, informacija, veličina);
   bufer[duljina - 1] = 0x0F;
   obrada_prekida_x();
   return 0;
}

void obrada_prekida_x()
{
   if (status == 1) { //slanje
      while (*RS && sljedeći < duljina) //ili samo: if (*RS)
         *RP = bufer[sljedeći++];       //             *RP = bufer[sljedeći++];
      if (sljedeći == duljina)
         status = 0; //slanje je gotovo
   }
}
```

5. Osmisliti i ostvariti raspoređivanje dretvi u jednostavnom sustavu koji koristi prioritetno raspoređivanje s 4 različita prioriteta (0-3). Definirati opisnik dretve sa `struct dretva` u koji treba dodati samo potrebne elemente za raspoređivanje te element `struct ostalo sve_ostalo`. Osmisliti strukturu podataka potrebnu za raspoređivanje te ostvariti funkcije:

```
void inicijaliziraj_raspoređivač();
void dodaj_u_pripravne(struct dretva *dretva);
void raspoređivanje();
```

Funkcija `dodaj_u_pripravne` se poziva kad se stvori nova dretva ili postojeća postaje pripravna (prethodno je bila blokirana), a `void raspoređivanje()` se poziva prije povratka u aktivnu dretvu, a čija je zadaća u globalnu varijablu `struct dretva *aktivna` postaviti kazaljku na aktivnu dretvu (odabrati najprioritetniju među pripravnim, uključujući i trenutno aktivnu) te po potrebi pozvati funkciju za zamjenu konteksta:

```
void zamijeni_kontekst(struct dretva *stara_aktivna, struct dretva *nova_aktivna);
```

koja postoji, dok argument `stara_aktivna` može biti i `NULL`. Pretpostaviti da će nakon inicijalizacije jezgra stvoriti bar jednu dretvu, dodati ju u red pripravnih i pozvati `raspoređivanje` te da će u nastavku rada u sustavu uvijek postojati barem jedna dretva spremna za izvođenje (npr. pored ostalih postoji i *idle* dretva koja se nikada ne blokira).

Rješenje:

```
struct dretva {
   int stanje; //0 nije pripravna, 1 je
   int prio;
   struct dretva *iduća;
   struct ostalo sve_ostalo;
}
struct dretva *aktivna;
#define BRPR 4
struct pripravne {
   struct dretva *prva, *zadnja;
}
pripravne[BRPR];

void inicijaliziraj_raspoređivač()
{
   aktivna = NULL;
   for (int i = 0; i < BRPR; i++)
      pripravne[i].prva = pripravne[i].zadnja = NULL;
}
void dodaj_u_pripravne(struct dretva *dretva)
{
   int i = dretva->prio;
   if (pripravne[i].prva)
      pripravne[i].zadnja->iduća = dretva;
   else
      pripravne[i].prva = dretva;
   pripravne[i].zadnja = dretva;
   dretva->iduća = NULL;
   dretva->stanje = 1;
}
void raspoređivanje()
{
   struct dretva *stara = aktivna, *prva = NULL;
   for (int i = BRPR - 1; i >= 0 && !prva; i--)
      if (pripravne[i].prva)
         prva = pripravne[i].prva;
   if (stara == NULL || stara->stanje == 0 || (prva != NULL && prva->prio > stara->prio)) {
      aktivna = prva;
      pripravne[prva->prio].prva = prva->iduća;//makni novu aktivnu iz reda pripravnih
      if (stara && stara->stanje)
         dodaj_u_pripravne(stara);

      zamijeni_kontekst(stara, aktivna);
   }
}
```

=> Moramo znati ovo za micanje aktivne dretve; da li je samo maknemo iz kazaljka aktivna, ili ju moramo i staviti u red pripravnih. Ako je prethodno ta dretva blokirana, kazaljka aktivna i dalje treba pokazivati na nju da bi u raspoređivanje mogli spremiti kontekst te dretve.

Red za pripravne može biti i jednostavna lista, uređena prema prioritetima. Tada je ubacivanje u listu linearne složenosti, ali u zadatku se nije tražilo da operacije budu konstantne složenosti.

Ovo treba biti zadnja radnja u ovoj funkciji ako se mijenja aktivna dretva

6. U sustavu koji koristi upravljanje memorijom na način da se u procesima koriste logičke adrese (npr. straničenjem) treba ostvariti jezgrinu funkciju `int sys_povećaj_buffer(void *stari, size_t veličina_prije, void **novi, size_t potrebna_veličina)`. Argument `stari` pokazuje na već dodijeljeni blok, argument `novi` pokazuje na varijablu u koju treba spremiti adresu novog povećanog bloka. Obje adrese su iskazane u logičkoj adresi koja se koristi unutar procesa. Funkcije koje se mogu koristiti (postoje unutar jezgre) su:

```
kprocess_t *dohvati_aktivni_proces();
void *pretvori_u_fizičku_adresu (kprocess_t *proces, void *logička_adresa)
void *pretvori_u_logičku_adresu (kprocess_t * proces, void *fizička_adresa)
void *zauzmi_blok_za_proces (kprocess_t *proces, size_t size);//vraća fizičku adresu
void oslobodi_blok_u_procesu (kprocess_t *proces, void *mem);//mem je fizička adresa
void memcpy (void *dest, void *src, size_t size); //obje adrese su fizička
```

Jezgrine funkcije pozivaju se mehanizmom prekida – funkciju `sys_povećaj_buffer` ne poziva dretva izravno već preko prekida. Unutar jezgrinih funkcija koriste se fizičke adrese.

Rješenje:

```
int sys_povećaj_buffer (void *stari, size_t veličina_prije, void **novi, size_t potrebna_veličina)
{
   void *fa_stari, **fa_novi, *novi_blok;

   kproces *proces = dohvati_aktivni_proces();
   fa_stari = pretvori_u_fizičku_adresu (proces, stari);

   novi_blok = zauzmi_blok_za_proces(proces, potrebna_veličina);
   memcpy(novi_blok, fa_stari, veličina_prije);
   oslobodi_blok_u_procesu(proces, fa_stari);

   fa_novi = pretvori_u_fizičku_adresu (proces, novi);//gdje staviti adresu novog bloka
   *fa_novi = pretvori_u_logičku_adresu(proces, novi_blok);//procesu vraćamo logičku adresu bloka

   return 0;
}
```

Završni ispit iz predmeta *Operacijski sustavi za ugrađena računala* 27. 6. 2022.

1. Izvorni kod jezgre za neki ugrađeni sustav nalazi se u direktoriju `kernel`, dok se programi nalaze u direktoriju `programs` (u direktoriju `kernel` se nalaze datoteke `io.c`, `core.c`, `sched.c`, `time.c`; u `programs`: `prog1.c`, `prog2.c`, `prog3.c`).

   a. (3) Napraviti skriptu za povezivanje `ldscript.ld` tako da se pripremi slika sustava koja će se posebnim alatom učitati u ROM koji je na adresi `0x10000`. Sliku pripremiti tako da se sve prvotno učita u ROM, ali da je za ispravan rad potrebno kopirati instrukcije jezgre, podatke jezgre uključujući konstante te podatke programa (bez konstanti) u RAM na adresi `0x50000`. Instrukcije programa i konstante koje program koristi trebaju se koristiti iz ROM-a (pripremiti za rad s tih adresa). Početna funkcija je `startup` (dodati `ENTRY(startup)` u skriptu).

ldscript.ld
```
ENTRY(startup)
ROM = 0x10000;
RAM = 0x100000;

SECTIONS{
     .rom_ram RAM : AT(ROM) {
          kernel* (*)
          programs* (.data .bss)
     }

     rom_ram_end = ROM + SIZEOF(.rom_ram);
     .rom_rom rom_ram_end : AT(rom_ram_end) {
          programs* (.text .rodata)
          //ili *(*) ALI NE MOŽE OVAKO NEŠTO GORE U PRVI ODJELJAK!
     }
     rom_rom_end = rom_ram_end + SIZEOF(.rom_rom);
}
```

   b. (2) Napisati `Makefile` za izgradnju slike sustava `slika.elf`. Mogu se koristiti i implicitna pravila te po potrebi i „recepti“. Pri povezivanju koristiti skriptu za povezivanje (dodati `-T ldscript.ld`). Za samo prevođenje pretpostaviti da su implicitne varijable već postavljene u datoteci `config.ini` koji treba uključiti na početku s `include config.ini`.

```
Makefile
include config.ini
IMG = slika.elf

OBJS = kernel/io.o kernel/core.o kernel/sched.o kernel/time.o \
     programs/prog1.o programs/prog2.o programs/prog3.o

$(IMG): $(OBJS)
     $(LD) -o $(IMG) $(OBJS) $(LDFLAGS) -T ldscript.ld $(LDLIBS)

#implicitna pravila su dovoljna za ostalo, ali može se npr. receptima:
#%.o: %.c
#    $(CC) -c $< $(CFLAGS)
```

2. (4) Neki procesor ima integriran sklop za prihvat prekida. On se programira na način da se u 32-bitovni registar `INT_EN` postavi broj čije jedinice predstavljaju omogućene prekidne ulaze (one koji se prihvaćaju mehanizmom prekida). Ulazi su numerirani od 0 do 31, gdje veći broj označava veći prioritet naprave. Kada naprava sklopu postavi zahtjev za prekid upiše se jedinica na odgovarajuće mjesto u registar `INT_RQ` (npr. naprava spojena za 7. ulaz postaviti će 7. bit). Po obradi prekida, potrebno je obrisati taj bit. Ako je prihvat prekida te naprave dozvoljen, zahtjev će se proslijediti procesoru, odmah, ako je ovaj zahtjev najvećeg prioriteta, ili kasnije, kad se zahtjev većeg prioriteta obradi. Procesor mora u registar `INT_CP` postaviti prioritet trenutne obrade, tj. odgovarajući bit tog registra treba biti postavljen u 1 (uspoređuje se samo najviše postavljeni bit). Procesor prihvaća prekid (automatsko ponašanje procesora pri prihvatu) tako da: 1. sprema sve registre procesora na stog te 2. u PC stavi adresu 0x10000 na koju treba postaviti prekidni program. Napisati prekidni program koji treba tamo postaviti uz pretpostavku da je sustav inicijaliziran, da se neki prekidi prihvaćaju a neki ne (što je programirano u registru `INT_EN`). Zahtjeve naprava koji ne izazivaju prekide, obraditi tek nakon obrade svih naprava koje izazivaju prekide (na kraju tih obrada provjeriti jesu li ostali samo ti zahtjevi i onda ih obraditi proizvoljnim redoslijedom). Pretpostaviti da su funkcije za obradu već postavljene u polje `hndl[]`. Dohvat najznačajnijeg postavljenog bita obaviti funkcijom (instrukcijom) `msb(x)`. Pomoćne operacije: postavi i-ti bit u 1: `X|=1<<i`; obrisi i-ti bit: `X&=^(1<<i)`.

```
prekidni_potprogram //na adresi 0x10000
{
   rq = msb(INT_RQ & INT_CP); //najprioritetniji zahtjev koji izaziva prekid
   INT_CP |= 1 << rq;         //postavi prioritet obrade u CP
   //“dozvoli_prekidanje“ nije potrebno jer ga procesor niti ne zabranjuje!
   hndl[rq]();
   INT_CP &= ^(1 << rq); //više se ne obrađuje ovaj prioritet
   INT_RQ &= ^(1 << rq); //zahtjev je obrađen, više ne čeka

   while (INT_RQ && (INT_RQ & INT_EN == 0)) {
        //ima zahtjeva koji čekaju, ali niti jedan od njih ne izaziva prekid
         rq = msb(INT_RQ); //proizvoljno, ovdje je uzet najvećeg prioriteta
         hndl[rq]();
         INT_RQ &= ^(1 << rq); //prekid obrađen
   }
}
```

3. (4) Neko ugrađeno računalo ima dva brojila: prvo BR1 odbrojava od 0xFFFFF do nule frekvencijom od 1 kHz, a drugo BR2 broji od 0 do 999 frekvencijom od 1 MHz. Prvo brojilo, kad dođe do nule izaziva prekid, dok drugo kad dođe do kraja ponovno kreće od nule (ne izaziva prekid). Oba brojila se mogu čitati i pisati. Ostvariti podsustav za upravljanje vremenom koji će imati sat u preciznosti 1 μs te jedan alarm u granulaciji 1 ms: `inicijaliziraj()`, `dohvati_sat()`, `postavi_sat(novi_sat)`, `postavi_alarm(kada, obrada)`, `prekid_BR1()`. Funkcije za dohvat i postavljanje sata koriste vrijeme u mikrosekundama (npr. 1234567890 μs, dok `kada` predstavlja apsolutno vrijeme, izraženo u milisekundama, npr. 1234567 ms).

```
sat = 0 //u mikrosekundama
obrada = NULL
kada = 0 //u milisekundama
MAX_BR1 = 0xFFFFF
učitano = MAX_BR1

inicijaliziraj () {
   sat = 0
   obrada = NULL
   kada = 0
   učitano = MAX_BR1
   BR1 = učitano
   BR2 = 0
}
```

```
postavi_sat(novi_sat) {
   sat = novi_sat
   kada = 0 //obriši alarm
   učitano = MAX_BR1
   BR1 = učitano
   BR2 = 0
}

dohvati_sat() {
   t = sat
   t += (učitano - BR1) * 1000 + BR2
   vrati t
   //ne ovdje mijenjati sat!!!
}
```

```
postavi_alarm(kada2, obrada2){
   kada = kada2
   obrada = obrada2
   sat += (učitano - BR1) * 1000 + BR2
   učitano = min(MAX_BR1, kada - sat/1000)
   BR1 = učitano
   BR2 = 0
}

prekid_BR1 {
   sat += učitano * 1000 //BR2 == 0 u tom trenutku!
   ako je (kada > 0) {
         ako je sat >= kada {
               kada = 0
               učitano = MAX_BR1
               BR1 = učitano
               obrada()
         }
         inače {
               učitano = min(MAX_BR1, kada - sat/1000)
               BR1 = učitano
         }
   }
}
```

4. (4) Neko ugrađeno računalo ima sporu bežičnu komunikaciju izvedenu posebnim sklopom koji za svaki prihvat 32-bitovna podatka generira prekid. Procesor tada mora prebaciti primljeni podatak iz registra sklopa na adresi `0xFFF30`, a da bi sklop mogao primiti sljedeći podatak. Pri slanju, podatak treba upisati na adresu `0xFFF40`. Ima li nepročitani novi podatak provjerava se preko statusnog registra na adresi `0xFFF50`, preko bita 0 (ako je postavljen). Slično, može li se poslati novi znak vidi se preko bita 1 (ako je postavljen). Odgovarajući bitovi statusna registra se automatski brišu nakon čitanja/slanja podataka. Sklop će generirati prekid i kad pošalje podatak, i kad može primiti sljedeći (na prekid treba provjeriti je li došao novi i/ili poslan idući). Podaci (na „višoj razini“) se šalju u paketima veličine do deset 32-bitovnih podataka omeđeni početnom i završnom vrijednošću (`0xF1F2F3F4` i `0x1F2F3F4F`) koji se računaju u veličinu paketa. Ostvariti upravljački program naprave, tj. ostvariti sučelja `init`, `send`, `recv`, `interrupt`. Funkcije `send` i `recv` se koriste za čitanje i slanje potpune informacije koju upravljački program treba zaštititi početnim i završnim brojem. Koristiti međuspremnike. U slučaju da prethodna poruka nije još poslana, idući `send` mora vratiti grešku (-1). Također, ako cijela poruka nije primljena, `recv` vraća -1. Odbaciti primljene podatke ako nisu započeli početnom vrijednošću. Slično ako prethodno primljen paket nije još pročitan s `recv`.

```
IN = 0xFFF30
OUT = 0xFFF40
STATUS = 0xFFF50
START = 0xF1F2F3F4
END = 0x1F2F3F4F
buf_in[10]          //10 brojeva (32-bitovnih)
buf_out[10]         //10 brojeva (32-bitovnih)
size_in = 0         //koliko je brojeva primljeno
size_out = 0        //koliko je brojeva poslano
size_to_send = 0    //koliko ukupno brojeva treba poslati

init() {
   *STATUS = 0
   size_in = 0
   size_out = 0
   size_to_send = 0
}
send(data, size) {
   ako je (size_to_send > 0)
         vrati -1 //prethodno nije još poslano

   buf_out[0] = START
   kopiraj(&buf_out[1], data, size)
   buf_out[size] = END
   size_to_send = 1 + size + 1
   size_out = 0

   ako je (*STATUS & 2 != 0) {
         *OUT = buf_out[size_out]
         size_out++
   }
}
```

```
recv(data, size) {
   ako je (size_in < 3 ILI buf_in[size_in - 1] != END)
         vrati -1

   size = size_in - 2
   kopiraj(data, &buf_in[1], size)
   size_in = 0
}

interrupt() {
   ako je (*STATUS & 2 != 0 I size_to_send > size_out) {
         *OUT = buf_out[size_out]
         size_out++
   }
   ako je (*STATUS & 1 != 0) {
         msg = *IN
         ako je (size_in < 10) {
               ako je ((size_in > 0 I buf_in[size_in] != END)
                        ILI (size_in == 0 I msg == START)) {
                     buf_in[size_in] = msg
                     size_in++
                     ako je (size_in == 10 I buf_in[9] != END)
                           size_in = 0 //odbaci podatke
               }
               //inače ignoriraj podatak
         }
   }
}
```

5. (4) Osmisliti i ostvariti raspoređivanje dretvi u jednostavnom sustavu koji koristi prioritetno raspoređivanje s 4 različita prioriteta (0-3). Definirati opisnik dretve i osmisliti strukturu podataka potrebnu za raspoređivanje (red pripravnih). Ostvariti funkcije za inicijalizaciju raspoređivača `void sched_init()`, za dohvat najprioritetnije dretve iz reda pripravnih `kthread_t *sched_get()`, za stavljanje dretve u red pripravnih `void sched_add(kthread *)`, za dohvat i micanje najprioritetnije dretve iz reda pripravnih `kthread_t *sched_pop()` te za odabir aktivne dretve (među trenutno aktivnom, ako takva postoji, i pripravnim dretvama) `void schedule()`. Raspoređivač neka radi prema prioritetu kao primarnom kriteriju te redu prispijeća kao sekundarnom (kao što to radi SCHED_FIFO). Dohvat opisnika trenutno aktivne dretve neka je moguć kroz `kthread_t *get_active()`, a postavljanje aktivne obaviti s `void set_active(kthread_t *)` (obje ove funkcije postoje, ne njih ostvarivati!). U `schedule()` dovoljno je na kraju postaviti aktivnu dretvu, povratak u nju će se ostvariti povratkom iz jezgrine funkcije – povratkom iz prekida (zato nije potrebno ništa drugo).

```
typedef struct kthread {
   int prio;
   struct kthread *next;
} kthread_t;

typedef struct rq {
```

```
   kthread_t *first;
   kthread_t *last;
} rq_t;

#define PRIO 4
rq_t rq[PRIO];

void sched_init() {
   int i;
   for (i = 0; i < PRIO; i++)
         rq[i].first = rq[i].last = NULL;
}
kthread_t *sched_get() {
   int i;
   for (i = PRIO-1; i >= 0; i++)
         if (rq[i].first)
               return rq[i].first;
   return NULL;
}
kthread_t *sched_pop() {
   int i;
   kthread_t *thread = NULL;
   for (i = PRIO-1; i >= 0; i++)
         if (rq[i].first) {
               thread = rq[i].first;
               rq[i].first = thread->next;
               if (rq[i].first == NULL)
                     rq[i].last = NULL;
               return thread;
         }
   return NULL;
}
void sched_add(kthread_t *thread) {
   if (rq[thread->prio].last == NULL) //nema tu dretvi
         rq[thread->prio].first = thread;
   else
         rq[thread->prio].last->next = thread;
   rq[thread->prio].last = thread;
   thread->next = NULL;
}
void schedule(){
   kthread_t *act = get_active();
   kthread_t *next = sched_get();
   if (!act || act->prio < next->prio) {
         next = sched_pop();
         set_active(next);
   }
}
```

6. (1) U sustavu koji koristi procese, tj. logički adresni prostor unutar njih, treba ostvariti jezgrinu funkciju `size_t sys_read(int fd, char *buf, size_t size)`, gdje je argument `buf` u logičkim adresama procesa. Pretpostaviti da se podaci interno u jezgri mogu dohvatiti iz datoteke/naprave funkcijom `kread()` s istim argumentima, ali gdje drugi argument mora biti fizička adresa. Skicirati ostvarenje te funkcije i navesti što je sve još potrebno od jezgre. Izostaviti provjeru argumenata, pretpostaviti da su oni već provjereni, da su prekidi zabranjeni i da će biti dozvoljeni nakon ove funkcije, prije povratka u dretvu (negdje drugdje).

```
size_t sys_read(int fd, char *buf, size_t size) {
   char *b = log_to_phy(buf); //potrebna je ta funkcija
   return kread(fd, b, size);
}
```